Turk J Elec Engin, VOL.15, NO.2 2007, © TÜBİTAK

Design and Implementation of an Ad-Hoc Routing Protocol for Mobile Robots[*]

Carlos AGÜERO, José M. CAÑAS, Miguel ORTUÑO, Vicente MATELLÁN
Robotics Group Universidad Rey Juan Carlos C/ Tulipán s/n, 28933 Móstoles, Madrid-SPAIN
e-mail: {caguero,jmplaza,mortuno,vmo}@gsyc.escet.urjc.es

Abstract

Mobile robots need to be able to communicate among themselves, as well as with hosts participating in the task that they are all involved in. Wired networks are obviously not suitable for mobile robots. Current wireless networks based on a fixed infrastructure (GSM, WiFi, etc.) to route packets may not be suitable because this infrastructure does not cover every place and the requirements of its resources. The best choice for mobile robots are Ad-Hoc networks, which are wireless and do not need a fixed infrastructure. This article describes PERA, a complete communications library including link, net, and transport layers for mobile robots with reduced communications capacity. The network layer is based on a well-known ad-hoc routing protocol adapted to limited devices. This protocol has been implemented and tested on EyeBot *mobile robots. Robots using PERA can send messages to other robots or hosts that are not directly reachable through their radio antenna range, by routing messages through intermediate mobile robots also running PERA. The design, implementation, testing and lessons learned in the development of PERA are presented in this article.*

Key Words: *Swarm communication, routing, and ad-hoc networks.*

1. Introduction

Communication ability is nowadays an essential component of any robot, both to let human users interact with them, and to let groups of robots communicate among themselves. Obviously, mobile robots require wireless technology. Wireless technology has been present in our lives since the mid-90's. WiFi (the popular trademark based on the WLAN/IEEE802.11 standard), BlueTooth, etc. are common today in everyday products. However, this type of technology is not adequate for limited mobile robots, as will be argued in following sections.

Most research efforts on distributed robotics have focused on foraging experiments, and distributed map building, where only link layer communications are used. For example, some works include communications as their main issue, for instance Rybski et al.'s work [12], only use communication among "visible"

[*]Work partially sponsored by grant DPI2004-07993-C03-01 by the Spanish Ministry of Education and Culture and grant and S-0505/DPI/0176 by Madrid Autonomical Community

robots in their foraging experiments; or use embedded networks, as in O'Hara & Balch's work [8], instead of the robot's capacity.

They all assume that any robot can communicate to another, which also assumes that the communication problem is really solved before their robot application may work. However, this assumption does not usually hold, in particular when it would be more useful, as for instance in catastrophic scenarios, as a collapsed building, or where external infrastructure has been severely damaged; or in others where the infrastructure is unavailable, as in the exploration of underground areas, for instance.

1.1. Limitations of current link layer technology

The major limitation of BlueTooth and WLAN/IEEE802.11 technologies is their range. The larger one is WLAN/IEEE802.11, which can reach up to a few hundred meters using omni-directional antennae in open spaces, though only 50-100m are actually achieved. This range can be enlarged by two methods:

1. Better technology. Technological improvements will certainly enlarge the range. However, these improvements have two major costs: energy requirements, and lower bandwidth when the amount of equipments grows.
2. Using the network layer service. Let us suppose a set of robots as presented in figure 1. Using just the link layer device, robot A can communicate to B. A network protocol will let robot A communicate to robot D by making intermediate robots re-send data. Link layer protocols (BlueTooth, WLAN/IEEE802.11, ...) have been designed only to communicate adjacent nodes (A and B).

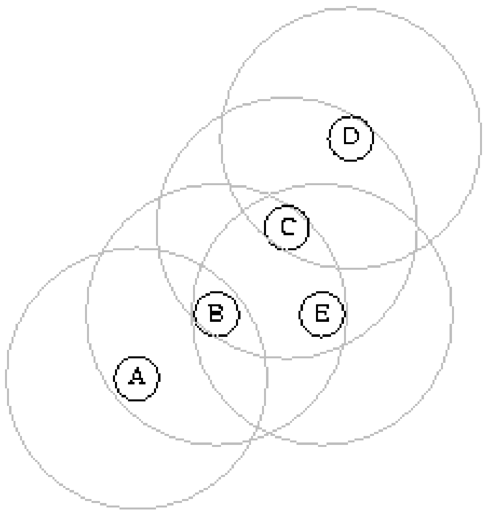

Figure 1. Ad-hoc network intuition

Adding network capacity to the robots expands the range of communications. However, some of the requirements of the standard network protocols may become unaffordable for robots of limited resources, as for instance Costbots [4], or micro-robots such as ANTS at MIT, or Millibots at CMU. Or the underlying layers (the link layer) may not provide compatible services: for example the radio link in the Eyebot cannot manage 4 byte addresses. The protocol presented in this paper deals with this problem.

Apart of the previous problems, traditional network protocols such as IP are not suitable for mobile robots. In particular IP networks (i.e., the internet) routing algorithms are based on the addresses, i.e., routing is based on the topology of the networks which is supposed to be fixed.

1.2. Ad-hoc networks

The robots in figure 1 may belong to a network in which all the nodes are robots. Each robot in this network will be able to send, route, and receive data packets. This is an *ad-hoc network*. This type of robot network can be self-deployed. There are no other network repeaters or access-points, just the robots infrastructure. This means that there would not be a single point of failure, which is very useful in many situations (collapsed buildings, or rescue situations in general [13]). Another advantage is the reduction of costs, economic or computational. One of the robots can be connected to another network (i.e., the internet), or to an expensive resource, such as a satellite link for instance, working as a gateway for all the other robots.

We can divide ad-hoc network protocols into two different groups: those based on *proactive* and those on *reactive* routing. Robots endowed with protocols based on proactive routing permanently update a table that allows them to route to *any* destination. This kind of protocol sends lots of routing information to adapt the table to changes in the network connectivity. Examples of this category of protocols include: DSDV (*The Destination-Sequenced Distance-Vector Routing Protocol*) [11], CGSR (*Clusterhead Gateway Switch Routing*) [14] and WRP (*The Wireless Routing Protocol*) [7].

On the other hand, protocols based on *reactive* routing only store the routes that have been really needed in their tables. When a robot wants to send a packet to an unknown destination, a route discovery process will be initiated on demand in order to learn such a new route. A route maintenance process is also needed to update the routes learned and to delete unused ones. Some examples of this category are: AODV (*Ad Hoc On-Demand Distance Vector Routing*) [9], DSR (*Dynamic Source Routing*) [6], LMR (*Lightweight Mobile Routing*) [2], TORA (*Temporary Ordered Routing Algorithm*) [9], ABR (*Associative-Based Routing*) [16] and SSR (*Signal Stability Routing*) [5].

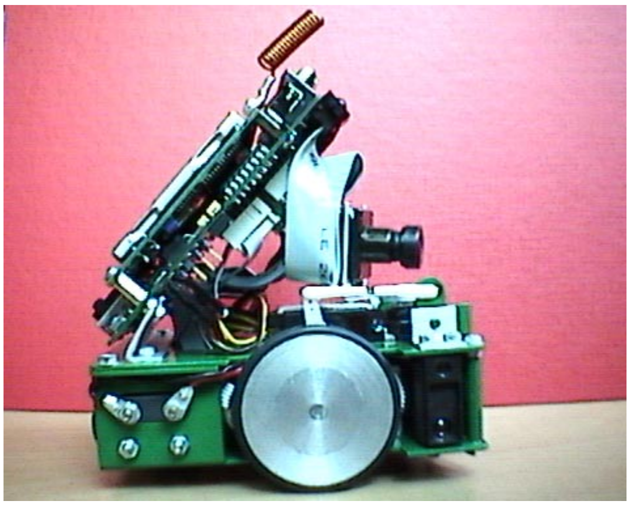

Figure 2. *EyeBot* robot

1.3. Robots with limited resources

Robotics hardware is being improved continuously, but there will always be simple robots: maybe because we need them to be cheap, disposable, . . . ; maybe because we need long operational times with severe power restrictions; or because their limitations may be a design requirement: robotic colonies, microrobots, nanorobots, toys, etc. All these reasons, and many others that we cannot envisage today, confirm that there will always be limited robots. There are several examples of this type of robot nowadays. Some of them are very popular in the robotics research community, such as the Khepera from K-Team [1], or Cotsbots [4], etc.

In this paper, we present the full description of PERA[2] which is an ad-hoc protocol designed to take into account the special requirements of small mobile robots. PERA is a protocol, specially designed for low cost robots, inspired in the AODV (*Ad-Hoc On-Demand Distance Vector*) [9] routing protocol. PERA is mainly a hierarchy of link, net, and transport protocols. The network protocol is the core of the library, and its multi-hop functionality is offered through a simple transport interface to let different applications running on a single robot use the communications facilities. PERA also specifies some constrains for the link layer, as for instance the non-blocking schema for receiving functions.

The implementation of PERA described in this paper has been tested on the *EyeBot* mobile robot (figure 2). *EyeBot* robots are endowed with three infrared sensors, two encoders and a camera. In addition, robots are supplied with a radio communication module that is used by PERA (the rudimentary antenna can be seen at the top of figure 2. All these devices are managed by the *RoBios* operating system. The PERA library is built using the *RoBios* API. This robot can be seen as a clear example of a limited robot: for instance the range of its radio system is very short, a few meters in the best environments, and it cannot carry a better system such as WLAN/IEEE802.11. The API has also severe limitations: the payload of the packets is very small (35 bytes), and its computer power is also very limited.

The remainder of this paper is organized as follows: the PERA protocol specification is presented in section 2. Section 3 describes the design of the PERA library. Finally, the implementation and the experiments made on the EyeBot mobile robots are described in section 4.

2. The PERA Ad-Hoc Routing Protocol

PERA has been designed to fulfill the following requirements:

- Every robot should be capable of sending data to any other robot.
- Every robot should be capable of receiving data from any other robot.
- The Movement of robots must not cause any disturbance to ongoing communications.
- Multiple applications running concurrently on the same robot can use PERA in order to send / receive data independently of each other.
- Every robot should be capable of sending data to a particular application running on a given robot (end-to-end communication).
- The library providing the PERA protocol should allow a choice between unicast and multicast transmission.

[1]http://www.k-team.com
[2]PERA stands for "Protocolo de Encaminamiento de Redes Ad-hoc" (Ad-hoc Routing Protocol).

It is not always possible to meet these requirements, in particular, when a path of intermediate robots between the source and the destination robot of a message is not available. No network protocol, including PERA, can help at this point.

Besides the previous requirements, which refer to the protocol itself, the implementation has been tested on *EyeBot* robots. The major constraints that *EyeBot* robots impose on PERA lie in the *RoBios* OS. For instance, the maximum size of a data packet is limited to 35 bytes when send between adjacent robots.

Some protocols based on reactive routing include the complete path in each data packet. This path is included in the packet when it is first sent, so that intermediate nodes can route the packet by consulting the path included in it. Due to the small size of data packets, this option was discarded.

PERA uses a protocol based on *on demand* routing that is table driven: each mobile robot maintains a table with routes. Packets only contain the address of the destination robot. Any intermediate robot which receives a packet which is not addressed to it, will consult its table in order to choose the next hop. Contrary to what happens in proactive protocols, as the ones used on the Internet, the protocol used in PERA only updates these tables on demand. Tables themselves are also created on demand.

The fields of the routing table that each robot manages in order to route packets received and whose final destination is another robot are shown in table 1. The main fields are the first three ones. The first one is the destination field. If the final destination of the packet does not appear in this column a new route discovery process has to be initiated. The second one is the next robot to which the packet has to be sent in order to finally reach the destination. The third one is incremented each time a new packet is sent in order to be able to distinguish between packets. The last three ones are used to keep the routes updated.

Destination	Next Hop	Sequence Number	Hop Count	Lifetime	Last Modified

Table 1. PERA routing table stored in each robot

Besides the use of the routing table to send packets, there are two main tasks that the routing protocol must solve: route discovery and route maintenance. They are used, respectively, to create an entry in the table when a packet must be routed to an unknown destination, and later, for keeping a given route updated in case it is still needed. In this way, each entry in the routing table can be erased or updated. The lifetime field, combined with the sequence number field ensure that a robot does not use old routes, and that routing cycles do not exist.

2.1. Route discovery

This process will be triggered by a robot when it wants to send a packet to another robot and there is no active route for the desired destination on its routing table.

The process is started by the sender that composes an RREQ (*Route Request*) packet, which includes the identifier of the sending robot and a locally generated RREQ ID. Together these will identify the request uniquely in the "net" made by the robots. Then, the robot broadcasts this message. Nearby robots that receive this RREQ must rebroadcast the packet. By flooding this initial message, the route discovery process ensures that the destination, in case it is reachable through any existing route, will be reached by this RREQ message.

All robots must check the ID and the origin of the RREQ in order to avoid loops and unnecessary flooding. In this way, an RREQ that has been previously received and resent by a given robot will not be sent again. The RREQ ID is incremented each time a new route discovery is initiated, so that when the conditions of connectivity change, a new route discovery will not be discarded by intermediate robots. Each time an intermediate robot receives an RREQ, it learns the reverse route to the source of the RREQ: the next hop to the original sender is the neighboring robot that has sent this RREQ to us (we assume symmetrical links).

If an RREQ is received by the final destination, that robot will send back an RREP (*Route Reply*) packet addressed towards the source of the RREQ received. This RREP packet will follow the reverse route that the RREQ made. This reverse route had already been learned and stored by all the intermediate routers when the RREQ was flooded. When the RREP reaches an intermediate robot, it learns the reverse route towards the origin of the RREP, and stores it on its routing table. Note that this is exactly the routing information that was originally searched for by the robot that initiated the route discovery.

An optimization that accelerates the pace of route discoveries is used in PERA. When a robot receives an RREQ, even if it is not addressed to it, it can reply with an RREP provided that it already knows a route to that destination (gratuitous RREP). The advantage of this hack is that RREQs do not need to be flooded everywhere in the net of robots, as long as someone already knows the final part of the route that is required. The main drawback to this solution is that outdated routes can be obtained.

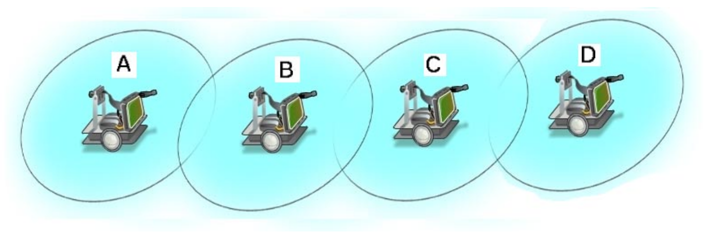

Figure 3. Route discovery

Let us consider a simple example to illustrate the process. For instance, consider the scenario of figure 3, where circles represent the radio range, and where all routing tables are initially empty. In this situation, robot A wants to send some information to robot D. First, A needs to discover a route towards D. It is thus necessary to initiate a route discovery process. Robot A creates an RREQ packet. This packet contains the source node address (A) and the current sequence number at node A, as well as the destination address (D). The RREQ also contains a broadcast ID (1), which is incremented each time the source robot initiates a new RREQ.

After creating the RREQ, robot A broadcasts the packet. When neighboring robot B receives it, it first checks whether it has seen this RREQ before, by checking the source address and broadcast ID pair. Each robot maintains a record of the source address / broadcast ID for each RREQ it receives.

In this example, robot B processes the packet. Robot B learns how to route packets to A and stores this information on its routing table. Then, robot B broadcasts the RREQ to its neighbors. This second RREQ is received by robot A, which silently discards the packet because it recognizes it as a packet already broadcast by it (in fact A was the originator of this RREQ). However, robot C, which also receives the RREQ sent by B, rebroadcasts it to its neighbors, after storing a new route towards A which passes through neighbour B, on its routing table. When robot B receives the RREQ broadcasted by C, it discards the packet.

When node D receives the packet broadcasted by C, it learns a route towards A, which passes through C. Then, it unicasts an RREP packet back to source A. Now, node D already knows to which neighbor it must send the RREP addressed to A, which is C. The RREP then reaches C, which in this way learns a route towards node D, and stores it on its routing table. Then, C checks its routing table and there it sees that the RREP addressed to A must be sent to B. When B receives the RREP, it learns a route towards D (through C) and stores it on its routing table. Finally, after checking its routing table, B sends the RREP directly to A. When A receives the RREP, it finally stores a route towards D (through neighbor B) on its routing table.

This concludes the discovery process in our example, once A has finally learned a route towards D. Now A can send the data to D, using a DATA packet type (see section 3 for the format of this packet). Note, that not only has A learned something, but intermediate robots have also learned new routes and they have also discovered new neighbors, etc.

A counter is added in each RREQ, to know the number of hops that a packet has followed. This counter is attached in each entry of a routing table, and when a robot rebroadcasts an RREQ, it must increment it. This counter is used to choose the shortest route. Only the number of hops is considered because time in every hop is considered a constant.

2.2. Route maintenance

All robots broadcast a *Hello* packet to inform their neighbors periodically that they are still in the vicinity. When a *Hello* packet is received, every route going through the source of that packet has its lifetime field updated. The *HELLO_INTERVAL* parameter establishes this period. The lifetime field is decremented as time passes by.

If a route is affected by the movement of an intermediate robot, an RERR (*Route Error packet*) packet will be sent towards the source of data in order to inform it that the route is no longer available. This RERR is sent by the robot that is one hop before the breakdown. A breakdown is considered when the lifetime field of the value of the routing table is not positive.

When a neighbor receives an RERR, it deletes its routes towards the unreachable robot, and then propagates the RERR backwards. When an RERR is received by the source robot, it initiates a new route discovery.

Each time a robot receives a broadcast, which may be a data packet, an RREQ, RREP, RERR, etc., from a given neighbor, it updates the lifetime field associated with that neighbor in its routing table. If at that time there is no entry for that robot in the table, the robot creates one.

3. Design of the PERA Library

We have built a communications library that can be used to send messages between any pair of robots in a herd, even when they are not directly reachable. It has been designed for robots with limited resources, in particular for the EyeBots. This functionality drastically increases the possibility of communication between EyeBots provided by the *RoBios* library.

The PERA library is structured in hierarchical modules following a traditional communications stack architecture. Each layer in the hierarchy provides services to the layer above, and uses services of the layer below through well defined interfaces. This design favors the porting among different types of robots.

In PERA we have considered four layers, ordered from lowest to highest in the hierarchy: link, net, transport and application, imitating the TCP/IP architecture. Each layer has an independent goal explained in the next subsections.

3.1. Link layer

The service this layer provides to the net layer is a transmission channel between neighboring robots directly reachable by its own radio. This layer is the only one that depends on the type of robot. If we want to use the PERA library with other robots (*Lego, Pioneer, ...*), other link layers, adapted to the physical communication channel of such robots, must be implemented.

The missions of this layer are to send and receive data to / from robots that are directly reachable through the EyeBot radio (for such a robot the range is about 1.5-2 m.).

The *EyeBot* operating system allows more than one application to run simultaneously in one robot (see section 3.3). Nevertheless, RoBios communication API is blocking. This forced us to develop a new non-blocking *receive* function in this layer in order to let more than one application use the communications hardware at the same time. This means that PERA offers an application to application communication mechanism, instead of a robot to robot one. This is very convenient in modern multi-threaded mini-robots.

3.2. Net Layer

The service offered by this layer to the transport layer is the routing of packets between any pair of robots, even if they are not neighbors, i.e., they are not in the same radio scope. This is the core layer of PERA and its main added value. It is here that we find the routing algorithms that PERA uses for route discovery and route maintenance, and where some data structures are implemented in order to store routing and control information.

Addressing

We have created an addressing scheme adapted to the peculiarities of the EyeBot communications infrastructure. Each robot must have a unique address. PERA uses one byte of each packet for this purpose, subdivided into three fields (see table 2).

Unic./ Mult.	*Host* address				*Port* number		
7	6	5	4	3	2	1	0

Table 2. Addressing scheme

The first field (bit 7), chooses between a unicast or multicast address. When bit 7 is set to 0, it specifies a unicast address and when it is set to 1 it specifies a multicast address, that is, an address that does not represent a single robot, but a set of them. The second field (bits 6-3) chooses the destination robot (we can address a maximum of 16 robots). Finally, the last field (bits 2-0) selects the port inside the destination robot. We allow up to 8 applications communicating at the same time on a single robot (see section 3.3).

Data packets format

PERA sets five types of messages to run all the protocols previously described. These packets are:

DATA : The DATA packet is made up by a header of 6 bytes (each one to a different field), and a variable length body for the data. The header fields are the message identifier (0), the address of the previous hop (1), the destination address of the data packet (3), the source address (4), and the size of the body data (5)

RREQ : The Route REQuest Packet (RREQ) described in the previous section is made up of the following 9 bytes: the message identifier (0), the address of the the previous hop (1), the number of hops from the source address to the robot that is currently handling the request (3), an identifier of the RREQ, that is unique for the *originator address* (4), the address of the destination robot (5); the last sequence number received by the source robot towards the destination (6); the address of the robot that originated the route request (7); and the current sequence number to be used for route entries.

RREP : The Route REply packet (RREP) packet contains eight bytes. The first one is the message identifier (byte 0), the address of the previous hop (1), the address of the next hop (2); the number of hops from destination address to the originator address (3); the address of the destination for which a route is being supplied (4); the sequence number associated with the route (5); the address of the source robot that issued the RREQ, that generated this reply (6); and a TTL (time to live) that is decremented at each hop, till reaching 0 when this packet be discarded (7).

RERR : The Route ERRor packet needs the following six bytes: an identifier (0); the address of the previous hop (1); the address of the next hop (2); the address of the robot that has become unreachable because of a link break (3); the last known sequence number associated with the unreachable robot (4); and the address of the destination robot towards which the RERR goes to.

HELLO : HELLO packets are periodically broadcast to keep neighbors informed that the robot is still close enough to send and receive data. This packet has just three fields: an identifier (0), the address of the next hop (1); and the current sequence number to be used for route entries pointing to (and generated by) the source of the route request.

3.3. Transport layer

The transport layer provides end-to-end communication by means of the abstraction of ports. It provides the service of multiplexing the radio channel among different applications running inside the robot. As previously stated, the use of ports allows application to application communication, instead of robot to robot.

In this way, it is necessary that an application *binds* itself to a free port when it wants to receive packets addressed to that port on that particular robot. Function *bind* associates one application to a port and does not allow two applications to listen to the same port. This function is part of the PERA API which will be detailed later.

Ports make it easier to program applications that are composed of different threads of control. For example, a thread can run a reactive controller which avoids obstacles by using infrared sensors, while another thread is running the code that guides the robot towards a ball using the camera. Imagine that another robot needs to send data to one of those threads on the first robot. Without ports it would be more difficult to do this task because we could not choose between threads.

Ports have been incorporated in the addressing scheme as shown in Table 2. Three bits have been reserved to identify the port, that is, eight different applications can be addressed in a robot.

Currently the PERA library does not provide any protocol on the application layer. In future releases we intend to provide application protocols adapted to the communication needs of the applications we run on our robots. In particular, we want to implement a subscription protocol that lets an application obtain periodic information, for example, sensor data.

4. Implementation and Evaluation

From the point of view of the PERA API, three functions have to be offered, which leads to some implementation decisions:

1. A non-blocking sending primitive is required, which has been solved by using a different thread for the PERA link layer.
2. Application addressing instead of robot addressing is also required, which has been solved by the addressing scheme that incorporates ports, as previously described.
3. It is desirable that the route discovery, route maintenance processes etc. are transparent to the user of the API. This requirement has been accomplished by providing a very simplified API.

The PERA API in the transport layer is made up of only three functions:

`bind(port)` which allows an application to listen, i.e. receive data, in the given port. Its main mission is to prevent other processes in the robot from using this port.

`recv(port, data)` which will return if new `data` received are in that `port`.

`send(destination, port, data)` , which lets an application send `data` to a `port` in the `destination`, a PERA valid address according to the addressing scheme previously explained.

In addition, the `init_PERA()` function has to be called before using the PERA API functions. This function initiates the PERA thread, and allocates data structures needed for the PERA operation.

The EyeBot operating system API (RoBios) imposed severe constraints on the implementation of the PERA library. RoBios is a proprietary code, which means that the source code is not available, so we had to use the communication API as it was. This API is the lower lever (link layer) in figure 4 and only provides robot to robot communication if the robots are adjacent, that is, if they fall inside their radio scope.

The net layer, which is the core of PERA, has been implemented using its own thread, represented by the big circle in the middle of figure 4. This thread checks each sending transport buffer, represented by blue boxes in figure 4 and, if there is some data pending stored on one of them by the transport layer, it sends them. This task is represented as a yellow box in the lower right part of the link layer in figure 4.

When the net thread receives data from the link layer, it stores them in the correct destination buffer, according to the destination port, (pink boxes in figure 4). This task is represented by the yellow L-box in the lower left corner of figure 4.

Other functions of the net layer are to route packets, that is, sending them to the right neighbor according to the routing table, and to answer RREP's received, in both cases it will use and update its routing table (red box in figure 4). Periodically, it also broadcasts a *Hello* packet.

The transport layer can just be seen as the sending and receiving buffers. Implementation of the `send` and `recv` functions only have to place data in appropriate buffers (`send`), read from them (`recv`), and

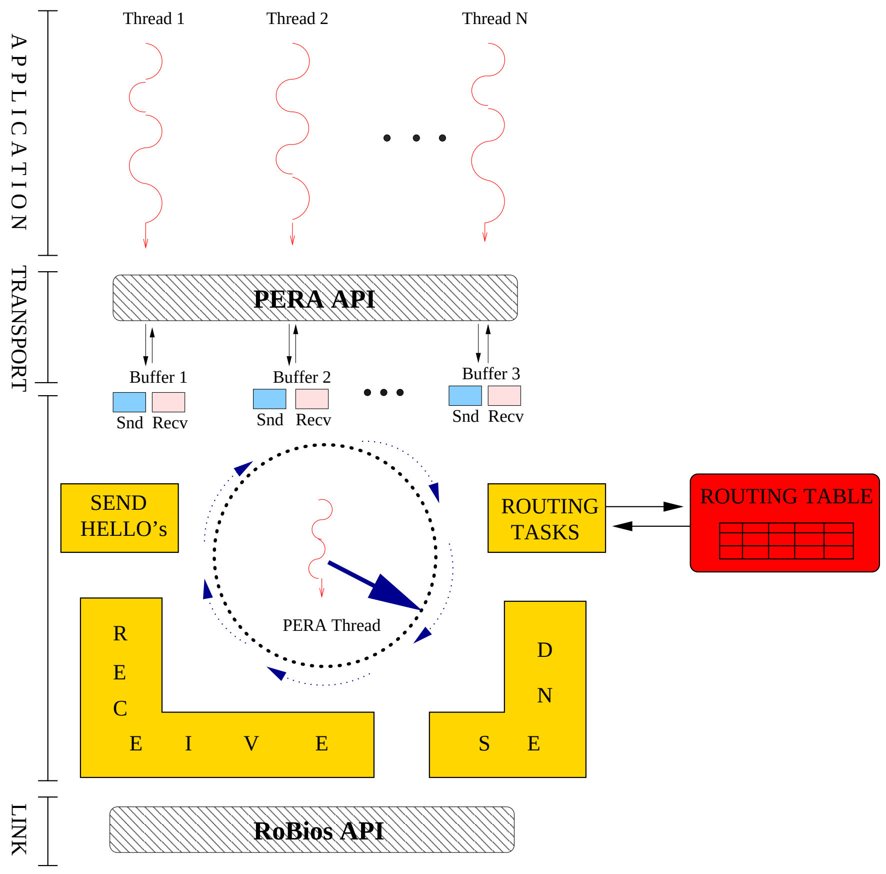

Figure 4. Implementation scheme using various layers

mark them as "in use" (`bind`), obviously using the classical concurrent protection mechanisms (semaphores provided by RoBios in this case) when accessing to share data from different threads. Applications will use PERA through those functions and buffers

Four experiments have been carried out to test the the performance of PERA in various scenarios. The most significant results are described in this section. For further information, the source code and detailed explanation of PERA can be obtained in the web of the URJC Robotics Group[3]. Protocol performance is measured with time relative to direct communication time. In all the experiments the *HELLO_INTERVAL* parameter has been set to a value of 20 seconds.

First we wanted to know the overhead introduced in the initialization of the communication systems (initialization of buffers, the link layer thread, etc.). Experiments have shown that there is a 15% increase in time when using PERA, and this increase is lineal to the number of robots. Initialization time depends on how many robots are powered on simultaneously (this is due to the underlying RoBios link protocols).

Then we wanted to know the performance in the discovery of new routes. The results once again show that the time needed grows lineally with the number of robots in the route, and so is proportional to the number of hops. Note that, for any new robot in the route two new messages have to be sent if that robot happens to be in the route. Note also that the inherent broadcast nature of robot radio communications,

[3]`http://www.robotica-urjc.es`

makes it independent from the number of robots that can be "seen" from the new one added. Finally, note that this linear increase refers to the number of packets through the first route, not in the whole net.

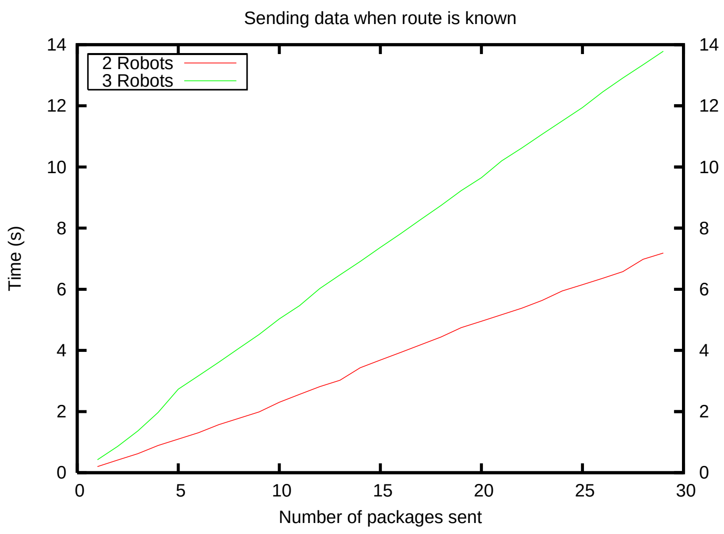

Figure 5. Time comparison when sending data through a known route

The performance of the protocol when sending data through a previously discovered route has also been tested. In figure 5 the results of a simple experiment are shown. First, the time that it takes for an increasing number of packets that two robots send to each other to be sent, when they can connect directly, is shown (red lower line). Then (green upper line), the same number of packets sent through another robot, in a three herd scenery, is shown. As previously stated, the time grows lineally with the number of intermediate hops the message has to cross. Absolute numbers are irrelevant and slow because of the underlying direct communication mechanism hardware.

Lastly, the recovery time when routes are lost (a relaying robot moves) has also been tested. Figure 6 shows the time that a sending robot needs to discover that the route is not working, till a new route is discovered. The set up of the experiment consisted of a simple 3 robots raw set up, where the middle one was duplicated, in fact there were 2 robots in the middle. The two robots placed in the middle were alternatively disconnected. The Time To Life (TTL) of the routes was set to 5 seconds, and the time or retransmission to one second in order that all the lost packets could generate a new route discovery. By looking at this figure, we can affirm that the reconfiguration time is almost the same as the route discovery time.

5. Conclusions and Further Work

The aim of the work presented in this paper was to provide an ad-hoc communications API for groups of reduced resources robots, which lets them communicate in environments where infrastructure is not available, or does not want to be used. We have proposed a modified communication protocol, an addressing scheme, and an implementation for the EyeBot robot. The implementation offers an alternative to the original robot operating system API for communications, which only allows direct communications.

The major improvements of this new API enable the possibility of sending data beyond the radio scope of the robot, i.e., the routing algorithm guarantees that if there were a route through other EyeBots,

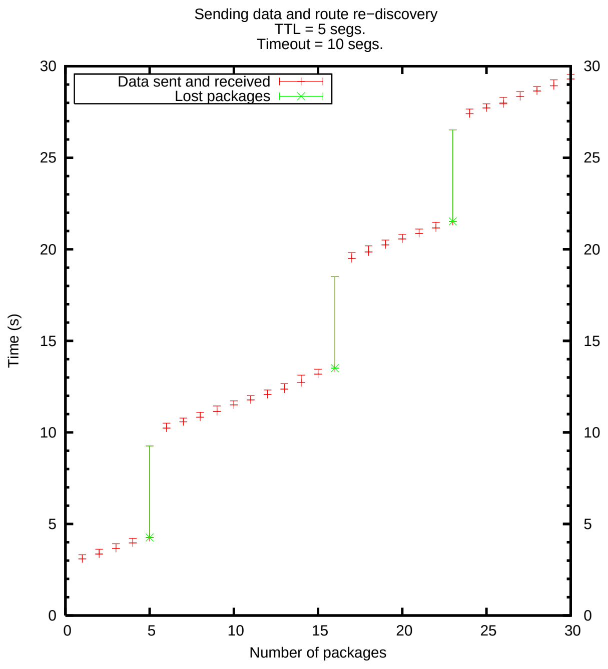

Figure 6. Route re-discovery times

data could be sent. Besides, the addressing scheme built allows communication among applications, not among robots, which lets an application send data to a particular program in a multiprocessing robot.

The main problem of this library is that there is a high risk of a packet being because PERA itself does not guarantee reliable communication on any of its layers. In this way, we could say that we have provided the equivalent to the UDP protocol, not to TCP. A place where message recovery could be provided is the transport layer, thus providing reliable transmission end-to-end by retransmission if it would be required. Another alternative is to implement recovery protocols at the link layer.

It would also be possible to use ACKs at the link layer in order to detect lost routes and to prevent transmission errors. This feature would discard false positives in the detection of lost routes, and would accelerate the recovery of lost messages in the case of transmission errors being the cause.

Finally, we are currently implementing a multicast extension, which is one sender and a group of receivers with the network scope (physical broadcast is currently provided by link level but only inside each radio scope). The addressing scheme of PERA already incorporates support for this kind of communication.

References

[1] T. Brunl, Embedded Robotics. Mobile Robot Design and Applications with Embedded Systems. Springer Verlag, 2003.

[2] M. S. Corson, A. Ephremides, “A Distributed Routing Algorithm for Mobile Wireless Networks”, *ACM/Baltzer Wireless Networks J.*, 1995.

[3] S. R. Das, C. E. Perkins, E. M. Royer, M. K. Marina, “Performance Comparison of Two On-demand Routing Protocols for Ad hoc Networks.” *IEEE Personal Communications Magazine special issue on Ad hoc Networking*, 2001.

[4] Sarah Bergbreiter, K.S.J. Pister. "CostsBots: An off-the-shelf platform for ditributed robotics". *Proc. of the 2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systmes (IROS)*, pp. 1632-1637. Las Vegas (USA), 2003.

[5] R. Dube, "Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks", *IEEE Pers. Commun.*. Volume 4, Number 1, pp. 36-45. 1997

[6] D. B. Johnson, D. A. Maltz, "Dynamic Source Routing in Ad-Hoc Wireless Networks", *Mobile Computing*, 1996.

[7] S. Murthy, J.J. Garca-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", *ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks*, 1996.

[8] K.J. O'Hara & T.R. Balch, "Pervasive Sensorless Networks for Cooperative Multi-Robot Tasks", Proc. 7th Int. Symp. Distributed Autonomous Robotic Systems, pp. 291-300. Toulouse, France, 2004.

[9] C. E. Perkins, Ad Hoc Networking, *Addison-Wesley*, 2001.

[10] C. E. Perkins, E. M. Belding-Royer, S. R. Das, "Mobile Ad Hoc Networking Working Group - Internet Draft", *http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-11.txt*, 2002.

[11] C. E. Perkins, P. Bhagwat, Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, *Comp. Commun*, 1994.

[12] P.E. Rybski, Amy Larson, Harini Veeraraghavan, Monica LaPoint, Maria Gini. Communication strategies in Multi-Robot Search and Retrieval: Experiences with MinDART. Proc. 7th Int. Symp. Distributed Autonomous Robotic Systems, pp. 301-310. Toulouse, France, 2004.

[13] RoboCup-Rescue Official Web Page, *http://www.r.cs.kobe-u.ac.jp/robocup-rescue/*

[14] E. M. Royer, C. Toh, "A Review of Current Routing Protocols for Ad-Hoc Movile Wireless Networks", *IEEE Personal Communications*, 1999.

[15] A. S. Tanenbaum, Computer Networks, 4th edition. *Prentice Hall*, 2002.

[16] C. Toh, A Novel Distributed Routing Protocol To Support Ad-Hoc Mobile Computing, *IEEE 15th Annual Int'l. Phoenix Conf. Comp. and Commun.*, 1996.